



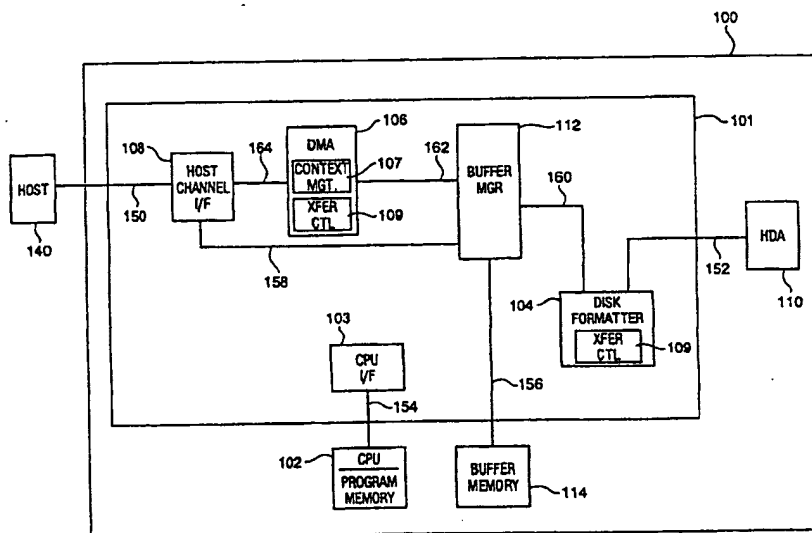
INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 3/06	A1	(11) International Publication Number: . WO 98/14861 (43) International Publication Date: 9 April 1998 (09.04.98)
(21) International Application Number: PCT/US97/17439 (22) International Filing Date: 29 September 1997 (29.09.97) (30) Priority Data: 08/724,385 1 October 1996 (01.10.96) US (71) Applicant: SYMBIOS, INC. [US/US]; 2001 Danfield Court, Fort Collins, CO 80525 (US). (72) Inventors: BORN, Richard, M.; 1909 Churchill Court, Fort Collins, CO 80525 (US). ELLIS, Jackson, L.; 2437 Sunstone Drive, Fort Collins, CO 80525 (US). SPRINGER, David, M.; 4304 Idledale Drive, Fort Collins, CO 80526 (US). NOELDNER, David, R.; 2300 West County Road 38E #119, Fort Collins, CO 80526 (US). WESTON-LEWIS, Graeme, M.; 903 Prescott Street, Fort Collins, CO 80525 (US). (74) Agent: LUCENTE, David, K.; Symbios, Inc., 2001 Danfield Court, Fort Collins, CO 80525 (US).		(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG). Published <i>With international search report.</i> <i>Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.</i>

(54) Title: METHOD AND STRUCTURE FOR MULTIPLE CONTEXTS IN STORAGE SUBSYSTEM TARGET DEVICES

(57) Abstract

A storage target device controller (101) (such as an embedded controller in an SCSI disk drive) processes multiple commands concurrently in accordance with the methods and structures of the present invention. Each command is stored within its own context within the target device controller (101) to retain all unique parameters required for the processing of each command. Processing of multiple commands permits switching of command contexts within the target device (100) to improve utilization of resources associated with the target device (100). For example, when a first, active, command context is prevented from further processing due to the status of the disk channel, an inactive command context may be swapped with the active command context to better utilize the host channel communication bandwidth. Similarly, a first active command context may be configured to automatically switch to a linked command context upon completion of processing to further ease management of multiple contexts. In a preferred embodiment of the present invention, a set of registers (214, 216, 218) contains the active context while a second set of registers (204, 206, 208) contains an inactive command context. The sets of registers (204, 206, 208, 214, 216, 218) are configured in such a way that the active and inactive context may be rapidly switched with no intervention by the microprocessor. The inactive register set (204, 206, 208) may be read or written directly by the microprocessor, or may be automatically loaded/stored from/to a buffer memory (114) in the target device (100) by shifting a predetermined context structure into the inactive register set (204, 206, 208) through an interface pad (212) with the buffer memory (114).



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

METHOD AND STRUCTURE FOR MULTIPLE CONTEXTS IN STORAGE SUBSYSTEM TARGET DEVICES

1. Field of the Invention

5 The present invention relates to storage subsystem target devices and, more specifically, relates to methods and associated structure for managing multiple command contexts within a target device by automatically switching between commands.

10 2. Discussion of Related Art

 A storage subsystem includes one or more storage devices for the persistent storage of information. The information and structure relating to the information storage is typically provided to the storage devices by a host computer system (or other storage management device) through an I/O
15 adapter. As used herein, initiator device refers to the host computer system and/or its associated I/O adapter. The target device, as used herein, refers to a selected one of the storage devices to which the initiator device directs an I/O request. The bus over which the initiator and target device communicate is referred to herein as the host channel.

20 The initiator/target device terminology is common to a standard bus known as Small Computer System Interface (hereinafter SCSI) - a popular electronic bus used to interconnect computers and peripheral devices. However, the problems known in the prior art discussed herein below and the solutions provided thereto by the present invention are common to any of
25 several well known host channel media and architectures, including but not limited to: Fibre Channel, SSA, IPI, IDE, EIDE, and others. For simplicity, the problems and solutions thereto addressed by the present invention are presented herein in the context of a SCSI host channel connection. However, one of ordinary skill in the art will readily recognize that the problems and
30 solutions addressed herein are applicable to other host channel environments.

 The SCSI bus has become a common choice for connecting computing devices (through a SCSI adapter in the computer system) to

storage peripheral devices such as disk drives, tape drives, disk arrays, and other storage and peripheral I/O devices. The specifications for the SCSI standards for the bus and the information exchanged thereon are publicly available in several forms including the ANSI publication **SCSI 2 Standards** 5 **Document X3T9.2/375R** revision 10L. Under the SCSI specifications, each device attached to the SCSI bus has a unique ID value by which it is identified on the SCSI bus. An initiator device initiates an I/O request to an identified target device and transfers any required data by asserting control over the SCSI bus (if available) and directing one or more commands to the identified 10 target device. In general, the target device performs the requested operation and returns information. The returned information may comprise simple sense or status information, or may comprise, for example, data retrieved from a mass storage peripheral device.

A disk drive stores and retrieves data via control circuits and methods 15 (an embedded controller) which command the head disk assembly (HDA) to perform the required physical storage and retrieval of information on the magnetic or optical medium. The interface within the controller to the HDA is referred to herein as the disk channel or formatter. Typically the data transfer bandwidth across the host channel is significantly greater than the bandwidth 20 on the disk channel.

A significant portion of the bandwidth of the host channel may be idle while the disk drive (or other SCSI peripheral device) attempts to process the I/O request via interaction with the actuator through the slower disk channel. Standard SCSI disconnect and reconnect protocols will permit other devices 25 to utilize the SCSI bus during this period. If no other initiator/target connection is established, the host channel will remain idle while the disconnected target device processes the received requests. Rather, the initiator simply awaits completion of the initiated plurality of commands (when the target re-selects the initiator).

30 Though the target device is incapable of responding to the first initiated command sequence, typical target devices also fail to respond to other commands which could be processed without requiring use of the busy disk

channel. In other words, present target devices process received commands serially rather than a plurality of commands concurrently. Target devices, in accordance with the SCSI standards, are often therefore referred to as single-threaded in the sense that they enforce a restriction that only a single
5 command (a single thread) can be processed at a given time. This single-threaded architecture may therefore tend to underutilize the high speed host channel and thereby extend the latency of other queued commands.

3. Summary of the Invention

10 The present invention solves the above and other problems, thereby advancing the state of the useful arts, by providing methods and associated structure to permit operation of multiple contexts within a target device. Though the disk channel may prevent the target from operating upon other commands which require interaction with the actuator, a wide variety of other
15 commands could be processed by the target device in accordance with the present invention. Each command being processed by a target device is referred to herein as a context. The methods of the present invention provide for storage within the target device of multiple contexts. One command may be in process within a first context while other commands may be
20 simultaneously serviced within other contexts of the target device.

Use of multiple contexts in accordance with the present invention allows processing of multiple commands with the target device to thereby better utilize the available bandwidth of the host channel and to reduce latency in servicing queued commands on initiator devices. For a disk device,
25 exemplary commands which may be processed in another context without requiring use of the disk channel are: read commands satisfied from a local cache, write caching, and non-data oriented commands and status (e.g., SCSI sense, inquiry, and status).

The methods and structure of the present invention provide for rapid
30 switching between multiple contexts that both require the disk channel to enable faster "back-to-back" (also referred to as coalesced) command processing. A first command may be in process utilizing the disk channel

while another command requiring the disk channel may be ready within the target device in a separate context. When the first context (command) completes, the second context is automatically switched into an active state to begin processing of the second command context. Such automatic
5 switching within the target device permits coalesced disk read or disk write operations to be processed without incurring penalties inherent in the rotational latency of the rotating disk storage medium.

This fast switching between command contexts within the target device provides significant flexibility to the systems architect in implementing difficult
10 command sequences. For example, a disk transfer can be managed within the target device simultaneously with a read command which requires only the return of cached data (a read cache hit).

Another aspect of the present invention is the automated management of the multiple context switching via a finite state machine model. The finite
15 state machine model of the present invention may be implemented in circuits independent of the microprocessor(s) used for overall target device control and communications. This finite state machine model therefore reduces the microprocessor overhead required for managing multiple contexts. Specifically, this aspect of the present invention provides for management of
20 multiple command contexts and automates the switching between them to improve utilization of disk and host channels and to reduce command processing latency on the host devices. Priorities are associated with each of the various states to determine whether and when to allow one command processing context (an inactive command context) to override the active
25 command context. When the priorities indicate that such a swap (or toggle) is appropriate, the active command context is rapidly swapped (toggled) with an inactive command context with no microprocessor interaction.

Still another aspect of the present invention provides for a particular structure of registers used in the processing of multiple contexts to avoid
30 stopping the disk channel due to unavailability of the host channel. Specifically in the preferred embodiment, a first set of counters associated with the active command context are used to control the exchange of data

with the target device while a second set of counters associated with the disk channel is used to store information about the exchange of disk blocks with the buffer memory while the host channel is occupied with another command context. Counters associated with the active command state in the first register set and with the disk channel activity are used to automatically determine when the active and inactive command structures should be swapped. A threshold comparison with counters associated with the disk channel is used to determine that the disk channel is about to pause if the host channel does not begin transferring data. In such a case, the active command context will be swapped with the inactive context associated with the disk channel at the next data block transfer boundary. The values saved in the counters associated with the disk channel are then used to adjust the counters of the active context when the active context has been swapped to the same command processing context as the disk channel.

It is therefore an object of the present invention to provide methods and associated apparatus for processing multiple contexts within a target device.

It is a further object of the present invention to provide methods and associated apparatus for processing multiple contexts within a target device so as to improve performance of the target device.

It is still a further object of the present invention to provide methods and associated apparatus for processing multiple contexts within a target device so as to improve performance of the target device as measured by utilization of the host channel bandwidth.

It is another object of the present invention to provide methods and associated apparatus for processing multiple contexts within a target device so as to improve performance of the target device as measured by utilization of the disk channel bandwidth.

It is still another object of the present invention to provide an apparatus in a target device for switching among a plurality of command contexts.

It is yet a further object of the present invention to provide an apparatus in a target device to toggle between two command processing contexts.

It is another object of the present invention to provide an apparatus in
5 a target device to switch between an active context and one or more inactive contexts for processing of commands within the target device.

The above and other objects, aspects, features, and advantages of the present invention will become apparent from the following description and the attached drawing.

10

4. Brief Description of the Drawings

FIG. 1 is a block diagram of a storage target device (disk) controller embodying the methods and structures of the present invention;

FIG. 2 is a block diagram of register sets for implementing the multiple
15 context management features of the present invention;

FIG. 3 is a flowchart describing the operation of the methods of the present invention to switch between multiple contexts to better utilize resources in the target device;

FIG. 4 is a block diagram of circuits of the present invention which
20 provide for rapid switching between a host channel activity and a disk channel activity in management of multiple contexts; and

FIG. 5 is a state diagram of the methods of the present invention which manage multiple contexts in accordance with the present invention.

25 5. Detailed Description of the Preferred Embodiments

While the invention is susceptible to various modifications and alternative forms, a specific embodiment thereof has been shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that it is not intended to limit the invention to the
30 particular form disclosed, but on the contrary, the invention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the invention as defined by the appended claims.

TARGET DEVICE CONTROLLER

FIG. 1 is a block diagram of a typical control component for a storage target device 100 (such as a disk drive) as presently known in the art. The target device 100 includes a controller 101 which provides control processing functions for communications with an attached host 140 via bus 150 and host channel interface 108. The disk formatter 104 within controller 101 performs real time control functions for the head disk assemblies 110 (HDA) via bus 152. As is known in the art, bus 150 may be any of several standard bus structure for attachment of a target storage device 100 including: SCSI, Fibre Channel, IDE, EIDE, IPI, PCI, SSA, etc. Bus 152 is representative of a bus structure which includes digital and servo control signals for operation of the actuators and motors as well as analog signals for the transfer of information between the controller 101 and the HDA 110 storage medium (such as rotating magnetic or optical disks).

As shown in FIG. 1, target device 100 includes CPU 102 which controls the overall operation of controller 101 through CPU interface 103 via bus 154. One of ordinary skill in the art will readily recognize that an interconnection bus (not shown) permits CPU 102 to exchange signals with the other components of controller 101. In addition, CPU 102 may be integrated with other components of the controller 101. Such design choices and considerations which determine the level of integration desired for a particular application are well known in the art. Buffer memory 114 in target device 100 is connected to buffer manager 112 in the controller 101 via bus 156. Buffer memory 114 is used to temporarily buffer, store, and retrieve information exchanged indirectly between an attached host 140 and the storage medium in HDA 110. The buffer memory 114 may also be integrated within components of the controller 101 as a normal matter of engineering design choice. Further, as shown in FIG. 1, CPU 102 often includes program memory for storage of instructions and variables for overall control of the controller 101.

The host channel bus 150 and host channel interface 108, in combination or alone, are referred to herein as a host channel. The disk formatter 104, bus 152, and HDA 110, in combination or alone, are referred to herein as the disk channel.

5 CPU 102 controls the overall operation of target device 100 in that it (in conjunction with host channel interface 108) receives requests from an attached host via the host channel, interprets the requests, and initiates appropriate processing within the target device 100 to satisfy the request (i.e., to read or write information on HDA 110). The disk channel, through disk
10 formatter 104, performs processing specific to the digital, analog, and servo controls required to manage HDA 110 via bus 152.

DMA 106 provides direct memory access between host channel interface 108 and buffer memory 114 via busses 156, 162, and 164 and buffer manager 112. The high bandwidth capability of the host channel
15 requires rapid access to the buffer memory 114 without the need for intervention by CPU 102. When such high speed data exchange is not required for processing of a particular command, host channel interface 108 may exchange information with buffer memory 114 via busses 158 and 156 and buffer manager 112 to thereby bypass use of DMA 106. Disk formatter
20 104 exchanges data with buffer memory 114 via busses 156 and 160 and buffer manager 112.

Context management 107, in conjunction with CPU 102, the host channel, and the disk channel, performs the specific processing of the methods of the present invention to manage multiple command contexts. In
25 particular, an active command context and one or more inactive command contexts are stored in structures within context management 107. As discussed in detail below, the context structures within context management 107 for storing contexts may be register sets or other memory components. Further, as noted below, additional command contexts may be stored in
30 buffer memory 114 to extend the number of inactive command contexts which may be managed by the methods of the present invention.

Logic within context management 107 determines when an inactive context must be loaded from buffer memory 114 into context management 107 or stored from context management 107 into buffer memory 114. In addition, context management 107 in conjunction with transfer control logic 109 (shown as contained within DMA 106 and disk formatter 104) determines when the active context and an inactive context within context management 107 should be swapped (toggled) in order to best utilize resources in the target device or to improve utilization of the host channel bandwidth.

One of ordinary skill in the art will readily recognize that FIG. 1 is intended only as a conceptual block diagram of a target device controller in which the methods and structure of the present invention may be advantageously applied. Many equivalent variants of the design shown in FIG. 1 may be apparent to those skilled in the art. In particular, the subcomponents of controller 101 may be logically subdivided in several functionally equivalent manners. In particular, one of ordinary skill may determine that particular functions may be implemented flexibly in programmed instructions within CPU 102 or may determine that the speed of custom integrated circuitry is required for management of multiple contexts. The methods of the present invention are therefore presented below as operable generally within the target device 100 of FIG. 1. The methods may be implemented as programmed instructions within CPU 102, or, as shown, as custom circuits and logic within the controller 101 (e.g., context control 107 and transfer control 109). The block diagram of FIG. 1 is therefore intended only to suggest one possible arrangement of components in a target device 100 in which the methods and structures of the present invention may be embodied.

COMMAND CONTEXTS

As used herein, context or command context refers to all information required to process a single command within the controller of the target device. Elements of such a context may include address and length values which are incremented or decremented as a requested data transfer

proceeds as well as flags which define particular options of the transfer. Such address pointers and counters refer to the host channel transfer of information to or from the disk channel and or buffer memory. As used herein, the active context is the command context of the command presently
5 being processed within the controller 101 of FIG. 1 on behalf of a command from an attached host device (i.e., the command presently transferring data over the host channel). The methods and structures of the present invention add the capability to manage one or more additional contexts within the target device. The additional contexts are also referred to herein as inactive
10 contexts or inactive command contexts.

Structures of the present invention store multiple command contexts for use in processing multiple commands. One active context and one or more inactive contexts are stored within the controller of the present invention. Methods of the present invention manage the stored contexts to
15 concurrently process multiple commands in the target device. Various conditions are monitored by the methods of the present invention to determine whether and when the active context should be replaced by one of the inactive contexts. Such a swap (also referred to herein as a toggle) of active and inactive contexts make the active context now inactive and starts
20 processing relating to a previously inactive command context (thereby making it the active context). The context information is rapidly and automatically switched by the structures and apparatus of the present invention to achieve the context switch with minimal delays imposed therebetween.

The controller of the target device therefore initially processes received
25 commands in the host channel interface. If the interface determines that the command can be processed immediately, a predefined context within the context management will be selected, loaded if necessary from the buffer memory, and swapped into the active context. The microprocessor of the controller generates contexts in anticipation of future commands received via
30 the host channel interface 108. In this manner, the microprocessor will not be the cause of a performance problem in most common command processing

circumstances, but rather performs background context management processing.

MULTIPLE CONTEXT MANAGEMENT METHODS

5 In general the methods of the present invention switch or swap contexts in order to improve particular performance attributes of the particular target device. For example, it may be desirable to maximize the use of the host channel bandwidth to move data to or from an attached host or minimize the latency in processing multiple commands from a host directed to one
10 target device. Or for example it may be desirable to assure minimum rotational latency in a disk storage target device so as to process back-to-back (coalesced) commands within the target device. These and other performance goals may be combined in making determinations as to which command context should be processed as the active context to maximize the
15 desired performance goals.

One of ordinary skill in the art will readily recognize that the particular performance goals are unique to the particular target device environment and application. The desired goals may vary according to the characteristics of the host channel, the disk channel, the buffer memory, and the HDA
20 assemblies. The methods of the present invention define a structure and methods in which any such performance goals may be measured and maximized through context switching between multiple commands contexts.

FIG. 3 is a flowchart describing at a high level the methods of the present invention to manage multiple contexts within the target device.
25 Detailed operation of the methods of the present invention is dependent upon the precise application environment and the performance goals to be improved. The processing of FIG. 3 is operable, as noted above, within controller 101 of FIG. 1 and more particularly within the combination of CPU 102, DMA 106, disk formatter 104, and buffer manager 112 of controller 101.
30 A specific state machine model for managing the switching of multiple contexts is presented in additional detail below with respect to FIG. 5. The

flowchart of FIG. 3 therefore presents the overall flow of control of the methods of the present invention.

Element 300 of FIG. 3 is first operable to initiate or continue processing the presently active command context. Element 302 is operable in response
5 to a change of state within the controller 101 of FIG. 1 to detect that processing of the active context is completed or paused. Changes of state that may initiate processing of element 302 include, for example, completion of the currently active command context, receipt of another command context from a host changes in the status of the disk channel, etc.

10 If element 302 determines that the present active context processing is complete, element 304 is next operable to determine if another presently inactive context is linked to the active (now completed) context. If such an inactive context is linked to the presently active context (just completed), that context is loaded into the active context by element 306. Element 300 is then
15 operable to initiate processing on the newly active context. The inactive context from which the active context is copied is then free to store another command context received from a host unless there remains status to be exchanged with the local microprocessor. In such a case the active context may remain swapped in the inactive context until the local microprocessor
20 clears the inactive context. Linking of contexts is noted below in additional detail as one advantageous application of the methods and structures of the present invention.

If element 304 finds no linked context associated with the completed active context, then element 308 is next operable to determine if there are
25 any inactive contexts presently storing commands in some stage of process (as distinguished from an empty, unused command context). If there are no inactive commands which may be actively processed, processing of the method of FIG. 3 is completed until another command is received from an attached host. Otherwise, element 310 is operable to load the active context
30 with the selected inactive context. If more than one inactive contexts are stored in accordance with the present invention, element 310 is operable to select a preferred one of the inactive contexts to be loaded into the active

context. This selection is in accordance with the needs and performance goals of a particular target device environment and application. In certain applications, the order of selection of an inactive context to become the next active context may be important. In others, the order of selection may be of
5 no significance. One of ordinary skill in the art will readily recognize many possible selection criteria among a plurality of inactive contexts as performed in element 310. Processing then continues by looping back to element 300 to initiate processing of the newly activated context. The inactive context from which the active context was loaded is then free for re-use by another
10 command received from the host, unless, as noted above, status information must be exchanged with the local microprocessor.

If element 302 determines that the presently active context is still in process, then element 312 is operable to determine if the target device resources are underutilized with respect to the particular performance goals
15 of the application of the present invention. If the target device resources are underutilized, element 314 is operable to select one of the inactive command contexts which will improve the resource utilization.

As noted above with respect to element 310, the criteria used to determine the desired performance goals are unique to a particular target
20 device application. For example, in many environments, increased utilization of the available host channel bandwidth is a desired goal as is reducing the latency in processing of host commands by accepting and processing additional command contexts. Likewise, reduction in delays in command processing within the target device due to rotational latency delays inherent in
25 rotating disk drives is a desirable goal (i.e., maintaining maximum utilization of the disk channel of the target device). The selection of an inactive context to be activated may therefore seek to improve these or other performance goals. Selection of an inactive context which is ready to transfer information over the host channel will improve host channel utilization. For example, a command
30 which will not require disk channel information exchange may operate to transfer information over the host channel regardless of the status of the disk channel. SCSI commands such as SENSE and INQUIRY are exemplary of

such commands which may transfer data over the host channel regardless of the state of the disk channel. An inactive command context which specifies a disk address near the present disk location as determined by the disk channel processor may be selected to reduce rotational latency delays. One of
5 ordinary skill will readily recognize many equivalent performance goals which may be improved by appropriate evaluation and selection by processing of elements 310 and 314.

Element 316 is then operable to swap the presently active command context with the selected inactive command context. The active context
10 thereby becomes inactive while the selected inactive context becomes active. The apparatus of the present invention, discussed in additional detail below, automatically performs the swap very quickly to maximize the potential performance benefits of activating the next command. Processing then continues with element 300 to continue or initiate processing of the newly
15 activated command context.

The methods of the present invention thereby continuously evaluate the performance of the target device with respect to predetermined performance criteria. The methods of the present invention then select a preferred one of a plurality of commands to process to thereby improve the
20 target device's performance.

CONTEXT STORAGE PREFERRED EMBODIMENT

FIG. 2 is a block diagram of circuits of the present invention which implement the context swap structure of the present invention in the best
25 presently known manner. The system depicted in FIG. 2 stores two contexts, each in a set of registers within context management 107 of FIG. 1. The active context 202 comprises a set of registers 214, 216, and 218 (also referred to herein as the active register set) which store the present values of parameters used in processing the presently active command (also referred
30 to herein as active parameters). The host channel interface and microprocessor of the controller may each manipulate the register values as required to initiate the presently active command. MUX's 234, 236, and 238

are associated with the registers 214, 236, and 238, respectively. Each MUX provides its output to its associated register via signal path 294, 296, and 298, respectively.

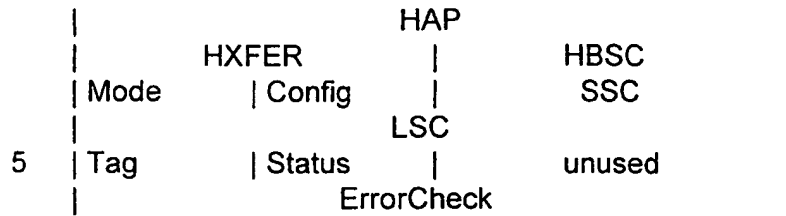
As the active command is processed, each register value is modified
 5 as required by the processing of the command. For example, address register values and counter register values are incremented and/or decremented as each datum is transferred from a source location to a destination location within the target device. The present value output of each register 214, 216, and 218 is applied via paths 254, 256, and 258,
 10 respectively, to a functional block 264, 266, and 268, respectively. The functional blocks adjust the register value as required for each datum transferred in processing of the command (i.e., incrementing or decrementing the values as required). One of ordinary skill will recognize that some values in a command context may require no adjustment with the transfer of each
 15 datum while other values may require more complex functions than simple increment or decrement operations. Such no-op or more complex functions are also represented by the function blocks 264-268.

The modified (or un-modified) output value of the functional blocks 264, 266, and 268 are applied to MUX's 234, 236, and 238, respectively, via
 20 paths 295, 297, and 299, respectively. The MUX's are controlled as required by other circuits (not shown) of the controller to apply the modified value input to the associated register.

The number of registers required (along with associated MUX and functional blocks) is determined in accordance with the number of parameters
 25 required to store a command context. Additionally, the width of the registers is determined by the register values required in the particular target device environment. In the best presently known mode of implementing the present invention, all registers are of equal width and the number of such registers is sufficient to store the following context data structures:

30

Byte3	Byte2	Byte1	Byte1
		HSBA	
		HSCA	



where:

- 10 HSBA - is the host segment base address (32 bits),
- HSCA - is the host segment ceiling address (32 bits),
- HAP - is the host address pointer (32 bits),
- HXFER - is the host block transfer count (16 bits),
- HBSC - is the host buffer space counter (16 bits),
- 15 MODE, CONFIG, TAG, and STATUS - are flags for the DMA operation (8 bits each),
- SSC - is the sector (block) size counter (16 bits),
- LSC - is the logical sector counter (32 bits), and
- ErrorCheck - is a check code to verify the success of reloading a context from
- 20 memory (32 bits such as a checksum or cyclic redundancy code).

One of ordinary skill will readily recognize that many equivalent data structures may be employed to represent the command context of a particular data transfer command. The particular address pointers and counters employed in the data structure above are therefore intended as exemplary of one mode of implementing a data structure for storage of a command context.

An inactive context 200 comprises a number of registers 204, 206, and 208 equal in number to that of the active context 202. Each inactive context register 204, 206, and 208 is of equal width and corresponds one-to-one with a register of the active context, namely registers 214, 216, and 218, respectively. The inactive register set 204, 206, and 208 are used to store parameters relevant to the processing of one inactive command (also referred to herein as inactive parameters). Each inactive register 204, 206, and 208 is associated with a MUX 224, 226, and 228, respectively, which determines the input value to be applied to the register via paths 284, 286, and 288, respectively.

The output of each inactive register 204, 206, and 208 is applied to an input of the MUX 234, 236, and 238, respectively, associated with its corresponding active register 214, 216, and 218, respectively, via paths 244,

246, and 248, respectively. Likewise the output value of each active register 214, 216, and 218 is applied to an input of the MUX 224, 226, and 228, respectively, associated with its corresponding inactive register 204, 206, and 208, respectively. In conjunction with the known settling times of such
5 register and MUX's, this structure permits the values of the active and inactive register sets to be rapidly swapped by appropriate selection of the MUX's 224, 234, 226, 236, 228, and 238.

As noted above, processing components in the controller can directly load and manipulate the values in the registers 204, 214, 206, 216, 208, and
10 218 via appropriate paths not shown. An exemplary path for such access to load values into the inactive registers is depicted as path 250 which applies a microprocessor supplied value to the input of the MUX's associated with the inactive registers. The more common access to the registers is via complete loading of the entire inactive register set with all values associated with an
15 inactive command context.

Since the entire inactive register set is typically loaded with a command context structure, an efficient path for such loading is provided. The entire inactive context (comprising another set of inactive parameters) is structured as a shift register wherein each register is chained to the next such that the
20 full width output value of one register is shifted to the input of the next register. The last register essentially shifts its value back to the first register to complete the shift chain. Specifically, the output value of inactive register 204 and 206 is applied via paths 244 and 246, respectively, to the MUX 226, and 228, respectively, of the next inactive register 206 and 208, respectively
25 in the inactive register chain. The last inactive register 208 applies its output to an input of MUX 210, via path 248, which in turn applies its output to PAD 212. PAD 212 also applies a value to path 292 as an input to MUX 224 for loading into register 204.

This structure allows the inactive register to act as a shift register chain
30 wherein each register shifts its value to the next register in the chain. Appropriate selection of the MUX's 224, 226, 228, and 210 allows the inactive registers to act as a shift register shifting each of the inactive register values,

in sequence, to the PAD 212 via output path 248. PAD 212 provides a direct interface for the inactive register chain to a buffer memory associated with the target device controller. In this manner, a complete inactive command context may be rapidly shifted out of the inactive context 200 to buffer
5 memory locations to thereby save an inactive command context. In like manner, a complete inactive command context may be shifted from buffer memory into the inactive context 200. The active command context may likewise be saved by swapping the active context 202 into the inactive context 200, then shifting the inactive context 200 out to buffer memory.

10 Additional inactive command contexts may therefore be stored in the buffer memory of the target device controller. One inactive context 200 may be rapidly swapped with the active context 202. Other inactive contexts may be stored in the buffer memory of the controller and then shifted into the inactive context 200 registers when required for swapping into the active
15 context 202.

The register structure depicted in FIG. 2 is exemplary of a structure which stores an active and inactive command context in a memory device (register sets) which permits rapid, automated, swapping (toggling) of the active and inactive contexts while also permitting the inactive context to be
20 loaded from, or stored to, a buffer memory. One of ordinary skill will recognize many equivalent structures suited to particular applications having particular context data structures associated therewith or particular functions associated with modifying the active context values. All such structures are intended as within the scope of the present invention.

25

DISK CONTEXT DUAL COUNTER OPERATION

While a plurality of command contexts may be managed as described above, the disk channel may continue to operate to transfer information on behalf of an inactive command context. For example, while a host generated
30 read request in the active context is awaiting receipt of required data for transfer to the host channel, an inactive command context may be activated (swapped with the active context) to utilize the idle host channel for other

transfers. The disk channel will continue to operate on behalf of the previously active (now inactive) read command. The inactive context will therefore not be updated to reflect the continued disk read transfers to the controller's buffer memory.

5 There are two exemplary situations where the disk block transfers must be held for later register updates (anytime disk and host contexts don't match). First, transfers on behalf of a presently inactive command context, and second, where a disk read ahead request reads more blocks than are to be transferred for a host command. Such a read ahead request may be
10 generated within the controller (by the CPU therein) in response to a host read command for a number of blocks and in anticipation of subsequent read commands received from an attached host requesting additional blocks in sequence. For example, though a host may request only a few blocks in an area of the disk, the controller may issue the read to the disk channel as a
15 command to read all blocks on the same track. As subsequent read commands are received from the host, those additional blocks (in the buffer memory cache) may be rapidly returned.

In this manner, the disk channel may be requested to transfer blocks to or from the disk in a manner which is independent of the counters in the
20 presently active command context. It is desirable to permit operation of the disk channel independent of the command context associated with the present host channel transfer (the active context). The present invention therefore utilizes a dual counter structure to efficiently manage this condition and permit the disk channel to continue operation regardless of the active
25 context presently utilizing the host channel.

The disk and host channels are each presumed herein to generate two pulsed signals for each block of data transferred to or from the HDA disk medium or host, respectively. Since both the host and disk channel utilize well known error checking and/or correction codes in the data to detect and/or
30 correct errors in the transferred data, a first pulse indicates that a block of data was transferred and a second pulsed signal indicates that the block was successfully transferred and is now "released" in the buffer memory of the

controller. Released as used herein means available for the next intended purpose such as processing by the controller or host or freed and available in the buffer memory for re-use by another command context. The first signal is referred to herein as an end of sector signal (EOS) and the second is referred to as a block released (BR). These signals as used in association with the host channel transfer are referred to herein as the host end of sector (HEOS) and the host block released (HBR) signals. With respect to the disk channel, the signals are referred to herein as the disk end of sector (DEOS) and disk block released (DBR) signals.

10 The command context on behalf of which the disk channel is presently operating (if any) is referred to herein as the disk context. The command context on behalf of which the host channel is presently operating is, as above, referred to herein as the active context. When the disk context is not the same as the active context, counters are used to count the blocks transferred and released for the disk channel to determine when the requested operation is completed. An independent (hence dual) set of counters is used to determine when a particular host transfer is completed. Certain of these counters are associated with the context data structure discussed above and are therefore saved and restored with the context as it is activated and deactivated. The dual counter feature of the present invention therefore maintains information regarding a count of blocks transferred by the disk channel which do not correspond to the presently active context. When the appropriate context is again activated (or created and activated in the case of a read ahead disk operation), then the stored block count is automatically applied to the counters of the command context to account for the pulsed signals missed in the active context.

25 The extra DBR pulses for a controller initiated read ahead (extra blocks beyond the host request which therefore do not correspond to a particular context) are counted in a counter referred to as the disk data block released counter (DDBRC). The extra DBR pulses are also stored in counters associated with the host channel, namely the host buffer space counter (HBSC) if the disk context is still the active context or the host data block

released counter (HDBRC) if the disk context is not presently the active context. The extra counts in the DDBRC indicate to the controller the status of the read ahead operation or to update the command context corresponding to the disk channel operation when it is again active.

5 If the next command context is a sequential read of the extra blocks read by the read ahead operation, the extra counts in the DDBRC and HDBRC may be used to rapidly, automatically adjust the other counters to account for the stored block pulse signals. If the extra block pulses were for a read ahead operation and the next host command does not require the use of
10 the read ahead blocks, the CPU of the controller may clear the extra pulse counts in the counter registers. If the extra pulse signals correspond to a command context that was temporarily inactive, the stored pulse counts are used to rapidly update that command context when the context is again activated.

15 FIG. 4 is a block diagram of an exemplary preferred embodiment of a circuit which maintains a count of disk transfers occurring through the disk channel on behalf of either a disk read ahead or on behalf of an inactive command context. The circuit of FIG. 4 represents a preferred implementation of the transfer control 109 elements of FIG. 1. As noted in
20 FIG. 1, the counter values may be integrated in part with the DMA 106 and in part with the disk formatter 104.

 A disk end of sector (DEOS) signal applied to path 478 is pulsed by the disk channel for each block (sector) which is transferred from the storage medium to the buffer memory of the controller. A disk block released (DBR)
25 signal applied to path 458 is pulsed by the disk channel for each block (sector) which is correctly transferred from the storage medium to the buffer memory of the controller. In like manner, a host end of sector (HEOS) signal and host block released (HBR) applied to paths 482 and 480, respectively, when a block is transferred to the host channel and successfully transferred
30 to the host channel, respectively.

 A context comparison element 400 generates a TRUE signal applied to path 450 (also labeled "A") if the currently active context corresponds to the

context which initiated the presently operating disk transfer (also referred to herein as the disk context). One of ordinary skill in the art will readily recognize that a context ID may be associated with each command context as one of the parameters stored in the context structure and registers. A
5 register associated with the disk channel may therefore retain the context ID of the command context which initiated the disk transfer. A simple comparator device may then determine if the presently active context ID matches the retained context ID within the disk channel.

Two counter elements are used under certain conditions to count the
10 number of DBR signal pulses: the HDBRC counter 404 and the DDBRC counter 410. The HDBRC counter 404 value is incremented once for each REL_HDBR pulsed signal applied to its input path 458 (each DBR signal pulse) and decremented once for each pulsed signal applied to its input path 454. The DDBRC counter 410 is incremented once for each DBR pulsed
15 signal applied to its input path 462 and decremented once for each REL_DDBR pulsed signal applied to its input path 464. The present value of HDBRC counter 404 is applied to element 408 via path 456 to determine if the value is not equal to zero. If the value on path 456 is not zero, element 408 applies a TRUE signal to path 452, otherwise a FALSE signal is applied
20 thereto.

Two additional counters are used to control the counter circuits. Disk transfer counter (DXFERC) counter 418 is decremented once for each DBR pulsed signal applied to its input path 474. The present value of DXFERC counter 418 is applied to element 424 via path 470 to determine if the value is
25 not equal to zero. If the value on path 470 is not zero, element 424 applies a TRUE signal to path 472, otherwise a FALSE signal is applied thereto. The host buffer space (HBSC) counter 406 is incremented once for each DBR pulsed signal applied to its input path 454. The HBSC 406 is representative of a register in the active context which indicates the number of sectors in the
30 buffer segment. The HBSC counter 406 is decremented by the host end of sector (HEOS) signal pulses applied to path 482 or is reset by the CPU of the controller when the context is reprogrammed for a new host command.

When processing a read ahead disk operation from the disk channel processor, the DXFERC counter 418 is loaded with the number of blocks to be transferred from the disk medium. The DXFERC counter 418 cannot be decremented further in response to DBR pulses on path 458 after it reaches
 5 zero. Rather, the additional DBR signal pulses are stored in the DDBRC counter 410 until the DXFERC counter 418 is again loaded with a non-zero value. At this time the stored DBR pulses in the DDBRC counter 410 are "released" to further decrement the DXFERC counter 418. The pulse count stored in the DDBRC counter 410 is decremented, "releasing" a DBR pulse to
 10 again decrement the DXFERC counter 418 until either the DDBRC counter 410 or the DXFERC counter 418 again reaches zero.

AND gate 416 receives the DDBR_NZ and DXFER_NZ signals on paths 468 and 472, respectively, to indicate whether the DDBRC counter 410 and the DXFERC counter 418, respectively, are presently zero. If both
 15 counters (410 and 418) are non-zero, AND gate 417 applies a TRUE value to path 464 indicating that a DBR signal pulse should now be released to decrement the DXFERC counter 418. AND gate 412 receives the DBR pulse on path 458 and the negated DXFER_NZ signal on path 472 and applies the logical and of the two signal as a pulse on path 462. The pulsed signal then
 20 increments the DDBRC counter 410 for each DBR signal pulse when the DXFER_NZ signal is FALSE (when DXFERC is equal to zero). AND gate 422 receives the DBR pulsed signal on path 458 and the DXFER_NZ signal on path 472 and applies the logical and of the two signals as a pulse on path 474. This pulsed signal and the REL_DDBR signal on path 464 are received
 25 as inputs at OR gate 420 which applies the logical or of the two signals as a pulse on path 476 to decrement the DXFERC counter 418.

The disk read ahead counter structure therefore operates in accordance with the following logic equations:

30 DDBR_NZ := not (DDBRC = 0)
 DXFER_NZ := not (DXFERC = 0)
 REL_DDBR := DDBR_NZ and DXFER_NZ
 increment DDBRC := DBR and (not DXFER_NZ)
 decrement DDBRC := REL_DDBR
 decrement DXFERC := REL_DDBR or (DBR and DXFER_NZ)

When processing a disk transfer function on behalf of an inactive command context (i.e., when the disk context ID is not the active context ID), the host buffer space (HBSC) counter 406 cannot be incremented in response to DBR signals until the disk context is again equal to the active context. Rather, the HDBRC counter 404 saves the DBR pulse signals in a count until the saved count can be released to the HBSC counter 406 when the disk context is again activated. When the disk context is again the active context, the saved count in the HDBRC counter 404 is rapidly transferred to the HBSC counter 406 as incrementing pulses.

In particular, AND gate 402 receives the HDBR_NZ signal on path 452 indicative of the DBRC counter 404 being non-zero (saved DBR pulses in the counter) and the disk context active signal on path 450 (also referred to herein as DCA). The logical and of the two signals is then applied to path 454 as a pulsed signal to increment the HBSC counter 406.

The inactive disk context counter structure is therefore operable in accordance with the following logic equations:

DCA := (disk context = active context)
 HDBR_NZ := not (HDBRC = 0)
 increment HBSC := REL_HDBR := DCA and HDBR_NZ
 decrement HBSC := HEOS
 increment HDBRC := DBR
 decrement HDBRC := REL_HDBR

This dual counter structure of the present invention enables efficient multiple context operation and allows simultaneous disk read ahead operation. The disk channel (on behalf of the disk context) can be reading ahead for one transfer operation while the host channel is occupied with a completely different operation. However, when the disk context and active context are again one and the same, the transfer counters can be rapidly, automatically updated to adjust for activity occurring while the active context was inactive. This rapid update occurs without requiring CPU intervention in the controller operation.

The dual counter structure has been described above with respect to read operations on the disk channel. One of ordinary skill will readily recognize that the structure is similarly functional to operate the disk channel to write data to the disk medium in parallel with other command contexts
5 operating the host channel. The disk channel continues to transfer data blocks from the buffer memory to the disk medium though the counter structure though the host channel is operating on a different active context. When the active context is again toggled back to the write command context corresponding to the disk channel operation (the disk context), the host
10 channel counter registers in the active context can be rapidly and automatically updated to reflect the disk channel activity.

Another feature of the dual counter structure depicted in FIG. 4 and described above is the use of a threshold value to determine when the disk channel requires host channel activity to maintain full utilization of the disk
15 channel bandwidth. For example, a read operation in the active context will initiate a corresponding read operation on the read channel. The active command context will typically be swapped (toggled) to another command to utilize the host channel while the disk channel begins reading requested blocks. When the disk channel (operating on the disk context) sufficiently fills
20 the buffer memory on behalf of the now inactive command context, the inactive command context must be re-activated to transfer the filled blocks to the host channel and thereby permit the blocks to be re-used for additional data blocks. If the inactive context is not swapped back to the active context, then the disk channel will be forced to stop data transfers until sufficient buffer
25 space is released by the host.

The dual counter structure of the present invention therefore includes a threshold comparison value for the disk channel operation. Referring again to FIG. 4, a disk buffer space (DBSC) counter 426 is incremented once for each DEOS signal pulse applied to path 478. The DBSC counter 426 is
30 decremented once for each pulse signal applied to path 484. The HBR signal pulse on path 480 (indicative of a successful transfer of a block to the host channel) is applied as an input to AND gate 430. The disk context active

(DCA) signal on path 450 (also labeled "A") is applied as the other input to AND gate 430. The logical and of the two signals (indicative of a successful host channel block transfer when the active context is the disk context) then generates output pulse signals applied to path 484.

5 The present output value of the DBSC counter 426 is applied as an input signal to threshold comparator 428 via path 486. The threshold comparator 428 compares its input value (the value of the DBSC counter 426) to a predetermined, programmable threshold value. If the DBSC counter 426 value is greater than or equal to the threshold comparator 428 predetermined
10 value, the comparator applies a logical TRUE signal to its output to thereby request the context management logic (107 of FIG. 1) to swap the disk context into the active context.

 The predetermined threshold value is chosen so as to permit the context management logic sufficient time to swap contexts before the disk
15 channel is forced to stop for lack of available buffer space. Specifically, the threshold must be a value such that the DMA transfer in process on behalf of the presently active context (if any) can complete its present block transfer. The specific operation of the context management logic is presented below in the form of a state machine. The use of the DBSC counter 426 to request a
20 toggle to the disk context is operable in accordance with the following logic equations:

increment DBSC := DEOS
decrement DBSC := HBR and DCA
DTCOG (request disk context toggle) := DBSC >= THRESH

25

CONTEXT MANAGEMENT STATE MACHINE MODEL

 The present invention also includes a state machine model for automated management of the multiple command contexts. This feature of the present invention further enables efficient management of the multiple
30 contexts within the target device while minimizing intervention required by the CPU. The state machine model of the present invention automatically controls the loading and storing of inactive contexts in the buffer memory from the inactive context register set (as discussed above with respect to FIG. 2).

In addition, the state machine model of the present invention automates the swapping of the active and inactive contexts as required to improve performance of the storage target device.

Any transfer request must be programmed into a context by the CPU
5 102 or the host channel interface 108 of the controller 101 of FIG. 1. There are preferably two contexts maintained in register sets within the context management logic 107 of the controller, namely the active context and (at least one) inactive context. The inactive context may be completely programmed by the CPU while the active context is busy transferring data as
10 requested.

The DMA 106 of FIG. 1 will transfer the requested data until it completes the transfer and then enter a paused state (setting appropriate status and interrupt conditions as required). The DMA 106 may also be placed in a paused state by external control such as by CPU 102 or by host
15 channel interface 108 or by the context management 107 within the DMA in order to change the context prior to completion of the active context. In addition, the DMA will pause when the HBSC counter (discussed above with respect to FIG. 4) counts down to zero indicating no further data is available (in the case of a read operation) or indicating the buffer is full (in the case of a
20 write operation).

While the DMA 107 is paused, for any reason, the active and inactive context may be swapped (toggled) such that the DMA will then resume operation in a different command context.

To perform a read or write operation the CPU 102 of controller 101
25 programs an inactive context into the inactive context register set. As noted above, the inactive context may be loaded by the CPU while the DMA operates on the active context. When the programmed inactive context is swapped with the active context, the DMA will then begin operation on the new read or write command. When the newly programmed command context
30 competes, the DXFERC counter (of FIG. 4) will be zero to thereby force any subsequent DBR pulse signals to accumulate in the DDBRC counter. If the next command may be coalesced with the just completed command, the

accumulated DBR pulses in the DDBRC counter will be released when the DXFER counter is programmed to the expected length of the next command. The next command may be coalesced if it follows sequentially from the previous command as indicated by the address and size of the two
5 commands.

In order to perform required context swaps automatically and quickly, the present invention provides for an auto-write context stored in the inactive context or in the buffer memory such that it can be quickly located and loaded into the inactive context (for swapping into the active context). The auto-write
10 context enables rapid initiation of a write command without requiring a host channel disconnect/reconnect sequence (such as in the SCSI protocols). If the command is stored in another context, such as the inactive, secondary, or disk contexts, then the host channel will be temporarily disconnected to permit the context to be programmed and stored. The auto-write context is
15 preprogrammed by the microprocessor with several parameters common to any write operation. Other values (such as the transfer length, or the LSC) are programmed directly by the host channel interface.

The host channel interface determines if a received write command is optimally processed in the auto-write context. If so, the host channel interface
20 remains connected to the host channel, parses the command and loads appropriate values into the auto-write context already loaded into the active context by the DMA context management logic. This operation requires no CPU intervention thereby minimizing the time between receiving the write command and actually transferring the requested data.

If the auto-write context is not ready for use by the host channel
25 interface or if the write command can be coalesced with another (earlier) write command, then the auto-write context is not used. Rather, the command will be either coalesced with another context or will be processed by the CPU to create another context for processing in the DMA. The CPU will program the
30 command into either the inactive context, the secondary context, or the disk context, depending upon the priority of the command relative to other contexts.

Another context, the secondary context, is programmed in buffer memory to prepare a second inactive context for later loading into the inactive context and eventual swapping with the active context. A read or write command which cannot use the auto-write context (and cannot be coalesced with another context), can be programmed by the CPU into the secondary context, the inactive context or the disk context. The inactive or secondary context is used when the related data can be stored in the buffer memory while the disk channel is occupied with other transfers. The CPU can program the new command directly into the disk context if the disk context is not presently in use corresponding to present disk channel activity. In the preferred embodiment, the new command would typically be programmed into the secondary context and moved to the disk context when the disk channel is available for additional transfer requests.

The secondary context is loaded into the inactive context and swapped to the active context when the host channel is idle (i.e., when the active and inactive contexts are not using the host channel). A read command in the secondary context, when activated, will return requested data from the buffer memory (read cache hits). A write command in the secondary context, when activated, will store received write data in the buffer memory. Later operations will be initiated to post the write data in the buffer memory to the disk channel for storage on the disk medium.

The state machine model of the present invention therefore controls the use of four contexts relating to host transfer requests, namely the active context, the inactive context, the auto-write context, and the secondary context. The disk context is the context which was the active context when the disk operation was initiated. As noted above, the active context may be swapped many times to service host channel transfers while the disk channel is operable on another context.

The state machine model of the context management logic automatically controls the swapping of the active and inactive contexts as well as the automated loading of the inactive context with a context (such as the auto-write context or secondary context). The state machine model operates

in accordance with a priority assigned to each context. The context management logic of the DMA element (of FIG. 1) will automatically load a new context (and swap it into the active context) if it is a higher priority than the current active context. The highest priority is the auto-write context
5 because the host channel interface has already determined that the write request must be satisfied immediately. Since the host channel interface remains connected to the host channel during this time, the DMA starts automatically on the new auto-write context as soon as possible.

The next priority is the context which corresponds to the disk context.
10 As noted above, when the DBSC counter reaches a pre-defined threshold value, the host context which corresponds to the disk context must be rapidly activated to avoid stopping the operation of the disk channel. If the disk context is already loaded into the inactive context, then the context management state machine will toggle (swap) the active and inactive contexts
15 as soon as possible. If an auto-write context request is pending at the same time that the disk context requires reactivation, the auto-write context will be activated for the duration of one block, then swap back to the disk context as the active context.

If the active context is not presently ready to transfer data and the disk
20 context does not presently require activation (i.e., the DBSC counter has not reached the pre-defined threshold value), then the state machine looks to the inactive context as the next priority. If the inactive context contains a valid, programmed context at the moment, then the active and inactive contexts are swapped (toggled) and the newly activated context will be resumed by the
25 DMA.

If the active context is idle, the disk context does not require activation, and the inactive context is idle (not programmed with a command context), then the secondary context is the last priority. The secondary context is a context programmed in the buffer memory (i.e., by the CPU in response to a
30 newly received host request), that is loaded into the inactive context, and swapped with the active context to commence operation thereon.

If there is no secondary context programmed in the buffer memory, then the disk context is swapped into the active context as a default condition to maintain the disk channel counter registers as up to date as possible.

Status bits associated with a context are used to determine if a context can be stored to buffer memory. If a context has outstanding status information, then the context will not be stored to buffer memory until the status information is detected, and cleared, by the CPU or host channel interface. This features allows the CPU to determine status without requiring access to the buffer memory.

The following table summarizes the priority of the various contexts for possible loading and swapping into the active context.

Priority	Description
1. Auto-write context	If the host channel has replied to a host write command such that data will be received therefrom imminently.
2. Disk context	DBSC register has reached a threshold that requires the host channel to release or provide additional buffer space to maintain disk channel activity.
3. Inactive context	The active context is idle and some command context is in the inactive context register set.
4. Secondary context	The active and inactive register sets are both idle.

The complete state machine model of the present invention incorporates operations required to load a context from the buffer memory into the inactive context and the subsequent operation to swap the loaded inactive context with the active context.

Specifically, the state machine model operates in accordance with the state diagram of FIG. 5. The state machine as depicted in FIG. 5 may be logically broken into two halves joined by the "idle" state 500. Transition 550 begins the processing of the state machine by entering the idle state 500 from the reset of the state machine. The states and transitions toward the left half of FIG. 5, namely states 502-508 and transitions 552-566, controls the swapping of active and inactive contexts. The states and transitions toward

the right half of FIG. 5, namely states 514-524 and transitions 572-590, control the automatic loading and storing of contexts in the buffer memory.

The state machine makes a transition from one state to another in response to particular logic states of a number of signals. Each transition
 5 between states is labeled with boolean conditions involving these variable (if any) which cause the state transition and also labeled with side effects affecting the values of these variable (if any) caused by the state transition.

The six transitions from the idle state 500 are mutually exclusive with respect to the conditions which trigger them. A priority value (1-7) is assigned
 10 to each transition from the idle state to resolve and conflicts in the transition is to be invoked at a given time. The priorities associated with the transitions 552-558 and 572-578 from the idle state 500 are preferably defined as follows for maximal improvement in the target device performance:

Transition	Priority
556	1
552	2,7
558	3
572	5
574	4
578	6

15 The priority assigned to transition 552 indicates that it is tested second and again seventh (last) to assure rapid toggle of the active context to the disk context from an active host (write) context.

With reference to FIG. 5, if a toggle is required to activate another context, the left side of the state machine model is operable to make a
 20 transition from the SWAP_IDLE state 500 to the appropriate TOGGLE_x state for the type of toggle requested. Specifically, and in order of priority indicated on FIG. 5, transition is made to state TOGGLE_WRITE 504 if an auto-write context is ready for operation, to state TOGGLE_DISK 502 if the disk context requires host channel processing, or to state TOGGLE_SECOND 506 if no
 25 other context is ready for activation. The TOGGLE_x state then prepares for a toggle to the corresponding context and makes a transition to the AUTO_TOGGLE state 508 to actually perform the required toggle between

the inactive and active contexts. Finally, the AUTO_TOGGLE state 508 make a transition back to the SWAP_IDLE state 500 when the swap has completed.

The toggle side of the state machine operates using the following state variables:

- 5 EN_TOG - 1 enables toggle state machine operation 0 disables.
- WTOG - 1 indicates a request to toggle to the auto-write context as active context.
- DTOG - 1 indicates a request to toggle to the disk context as active context.
- 10 ITOG - 1 indicates a request to toggle to the inactive context
- STOG - 1 indicates a request to toggle to the secondary context
- WPRIV - 1 indicates the auto-write context must be held in the active context.
- DPRIV - 1 indicates the disk context has priority
- 15 DEFAULT_TO_DISK - 1 indicates that the disk context is to be checked again last as the default priority when no other context requires activation.
- AUTO_PAUSE - 1 indicates a request to pause the DMA transfer from the disk channel to memory at the next sector boundary.
- AUTO_PAUSED - 1 indicates that the DMA has reached the next sector
- 20 boundary and has paused.

The toggle (auto-swap or left side) portion of the state machine then operates in accordance with the following state transition table.

State (label)	Condition (label)	Next State (label)	Side Effect
SWAP_IDLE (500)	EN_TOG=1 and (DTOG=1 or DEFAULT_TO_DISK=1) (552)	TOGGLE_DISK (502)	AUTO_PAUSE := 1
SWAP_IDLE (500)	EN_TOG=1 and WTOG=1 (556)	TOGGLE_WRITE (504)	AUTO_PAUSE := 1
SWAP_IDLE (500)	EN_TOG=1 and (STOG=1 or ITOG=1) and DTOG=0 and DPRIV=0 and WPRIV=0 (558)	TOGGLE_SECOND (506)	AUTO_PAUSE := 1
TOGGLE_DISK (502)	AUTO_PAUSED=1 (560)	AUTO_TOGGLE (508)	AUTO_PAUSE := 0
TOGGLE_WRITE (504)	AUTO_PAUSED=1 (562)	AUTO_TOGGLE (508)	AUTO_PAUSE := 0
TOGGLE_SECON D (506)	AUTO_PAUSED=1 (564)	AUTO_TOGGLE (508)	AUTO_PAUSE := 0
AUTO_TOGGLE (508)	true (566)	SWAP_IDLE (500)	

With reference to FIG. 5, if no toggle is required, then the lesser priority load/store operations are processed if required by the right side of the state machine model of FIG. 5. In particular, the state machine model is operable to make a transition from the SWAP_IDLE state 500 to the appropriate STORE_x state for the type of load/store operation requested. Specifically, and in order of priority indicated on FIG. 5, transition is made to the STORE_WRITE state 516 if an auto-write context is ready for loading to the inactive context, to the STORE_DISK 514 if the disk context is to be loaded into the inactive context, or to the STORE_SECOND 518 if the secondary context is to be loaded into the inactive context. The WAIT_STORE state 520 then stores the present inactive context into buffer memory to prepare for loading a new inactive context. The WAIT_STORE state 520 then transitions to the LOAD state 522 after the present inactive context is stored and the new context can be loaded. The WAIT_LOAD state 522 then awaits the completion of the loading of the new inactive context from buffer memory. Finally, the WAIT_LOAD state 522 make a transition back to the SWAP_IDLE state 500 when the store/load operation has completed.

The load/store side (or right side) of the state machine operates using some of the same state variable listed above and the following additional state variables:

- 5 EN_AUTO - 1 indicates enabled 0 indicates disabled for load/store operation.
- SLOAD - 1 indicates that a load of the secondary context into the inactive context is desired.
- DLOAD - 1 indicates that a load of the disk context into the inactive context is desired.
- 10 WLOAD - 1 indicates that a load of the auto-write context into the inactive context is desired.
- CNTXTVLD = 1 indicates the context is presently valid.
- CNTXTDID - the value of the context ID which is presently the disk context.
- SCONTXT - the ID value of the secondary context.
- 15 WCONTXT - the ID value of the auto-write context.
- AUTO_CTXSTORE - 1 indicates a request to store the present inactive context in buffer memory for later re-loading.
- AUTO_CTXLOAD - 1 indicates a request to load the new context from buffer memory into the inactive context.
- 20 AUTO_CTXID - the ID value of the new context to be loaded.

The load/store (or right side) portion of the state machine then operates in accordance with the following state transition table.

State (label)	Condition (label)	Next State (label)	Side Effect
SWAP_IDLE (500)	EN_AUTO=1 AND DLOAD=1 (572)	STORE_DISK (514)	AUTO_CTXSTO RE := 1
SWAP_IDLE (500)	EN_AUTO=1 AND WLOAD=1 (574)	STORE_WRITE (516)	AUTO_CTXSTO RE := 1
SWAP_IDLE (500)	EN_AUTO=1 AND SLOAD=1 AND GET_WCONTXT=0 AND DPRIV=0 (578)	STORE_SECOND (518)	AUTO_CTXSTO RE := 1
STORE_DISK (514)	CNTXTVLD=0 (580)	WAIT_STORE (520)	AUTO_CTXSTO RE := 0, AUTO_CNTXTID := DCONTXT
STORE_WRITE (516)	CNTXTVLD=0 (582)	WAIT_STORE (520)	AUTO_CTXSTO RE := 0, AUTO_CNTXTID := WCONTXT
STORE_SECON D (518)	CNTXTVLD=0 (584)	WAIT_STORE (520)	AUTO_CTXSTO RE := 0, AUTO_CNTXTID := SCONTXT
WAIT_STORE (520)	CNTXTVLD=1 (586)	LOAD (522)	AUTO_CTXLOA D := 1

LOAD (522)	CNTXTVLD=0 (588)	WAIT_LOAD (524)	AUTO_CTXLOA D := 0
WAIT_LOAD (524)	CNTXTVLD=1 (590)	SWAP_IDLE (500)	AUTO_CTXLOA D := 0

The state machine model described above may be implemented in any of several well known forms, including software in a general purpose computer, programmable logic devices, or custom discrete or integrated circuits. As described herein, the state machine model is preferably implemented in custom integrated circuits integrated as shown in FIG. 1 with other components in the controller 101 of a target device 100. One of ordinary skill in the art will readily recognize that other state variables and intermediate states may be required to implement a state machine in a particular integrated circuit medium with particular timing and speed constraints. Design tools well known to those skilled in the art typically aid the designer in the creation and insertion of such timing related states and transitions.

While the invention has been illustrated and described in detail in the drawings and foregoing description, such illustration and description is to be considered as exemplary and not restrictive in character, it being understood that only the preferred embodiment and minor variants thereof have been shown and described and that all changes and modifications that come within the spirit of the invention are desired to be protected.

CLAIMS

What is claimed is:

1. In a storage target device having a host channel and context storage
5 means for storing the context of a plurality of commands, a method for improving utilization of resources of said target device comprising the steps of:
- processing an active command in an active context in said context
storage means;
- 10 storing at least one inactive command in at least one inactive context
in said context storage means;
- determining that processing of another command selected from said at
least one inactive command would better utilize said resources of said target
device; and
- 15 swapping said active context and one of said at least one inactive
context to process said another command in response to said determination
that processing of said another command would better utilize said resources
of said target device.
- 20 2. The method of claim 1
- wherein the step of determining includes the step of determining that
the communication bandwidth of said host channel is underutilized, and
- wherein said swapping step is responsive to said determination that
said communication bandwidth of said host channel is underutilized.
- 25 3. The method of claim 1
- wherein the step of determining includes the step of determining that
processing of said active command has completed, and
- wherein said swapping step is responsive to said determination that
- 30 said processing of said active command is completed.

4. The method of claim 3 wherein a command context includes information linking said active command to a next command of said at least one inactive command and wherein the step of swapping includes:

5 copying the context of said next command to said active context; and
initiating processing of said next command.

5. The method of claim 1 wherein said context storage means includes storage for a plurality of inactive contexts for storage of inactive commands and wherein the step of swapping includes the steps of:

10 selecting one of said plurality of inactive commands stored in one of said plurality of inactive contexts; and

swapping said active context and said inactive context to process the selected inactive command in response to said determination that processing of said second command would better utilize said resources of said target
15 device.

6. The method of claim 5 wherein said active context and said plurality of inactive contexts are stored in a buffer memory associated with said storage target device.

20

7. The method of claim 5 wherein said storage target device includes an active register set for storage of said active context and wherein said storage target device further includes at least one inactive register set for storage of at least one of said plurality of inactive contexts.

25

8. The method of claim 7 wherein said storage target device includes a plurality of inactive register sets for storage of said plurality of inactive contexts.

30 9. Apparatus in a storage target device controller to provide multiple contexts for concurrently performing a plurality of commands generated by an initiator device, said apparatus comprising:

4. The method of claim 3 wherein a command context includes information linking said active command to a next command of said at least one inactive command and wherein the step of swapping includes:

5 copying the context of said next command to said active context; and
initiating processing of said next command.

5. The method of claim 1 wherein said context storage means includes storage for a plurality of inactive contexts for storage of inactive commands and wherein the step of swapping includes the steps of:

10 selecting one of said plurality of inactive commands stored in one of said plurality of inactive contexts; and

swapping said active context and said inactive context to process the selected inactive command in response to said determination that processing of said second command would better utilize said resources of said target device.
15

6. The method of claim 5 wherein said active context and said plurality of inactive contexts are stored in a buffer memory associated with said storage target device.

20

7. The method of claim 5 wherein said storage target device includes an active register set for storage of said active context and wherein said storage target device further includes at least one inactive register set for storage of at least one of said plurality of inactive contexts.

25

8. The method of claim 7 wherein said storage target device includes a plurality of inactive register sets for storage of said plurality of inactive contexts.

30 9. Apparatus in a storage target device controller to provide multiple contexts for concurrently performing a plurality of commands generated by an initiator device, said apparatus comprising:

an active context storage means for storing active parameters relevant to the performance of an active one of said plurality commands;

at least one inactive context storage means for storing inactive parameters relevant to the performance of at least one inactive command of
5 said plurality of commands; and

swapping means for swapping the active parameters in said active context storage means with the inactive parameters stored in said at least one inactive context storage means.

10 10. The apparatus of claim 9 further comprising:

a buffer memory having segments wherein one of said segments corresponds to said active context storage means and wherein each of said at least one inactive context storage means corresponds to one of said segments.

15

11. The apparatus of claim 9:

wherein said active context storage means includes an active register set for storage of said active parameters; and

wherein said at least one inactive context storage means includes at
20 least one inactive register set for storage of said inactive parameters.

12. The apparatus of claim 11 wherein each of said at least one inactive context storage means corresponds to a unique set of said at least one inactive register set.

25

13. The apparatus of claim 11 wherein said at least one inactive register set comprises a single inactive register set and wherein said apparatus further comprises:

a buffer memory for storage of additional inactive contexts of said at
30 least one inactive context;

storing means for storing said single inactive register set into an identified one of said additional inactive contexts; and

loading means for loading an identified one of said additional inactive contexts into said single inactive register set.

14. The apparatus of claim 13 wherein said loading means comprises:
- 5 means for sequentially loading each of said inactive parameters of said identified one of said additional inactive contexts into one register of said single inactive register set; and
- means for shifting each loaded inactive parameter from said one register to an appropriate register in said single inactive register set.

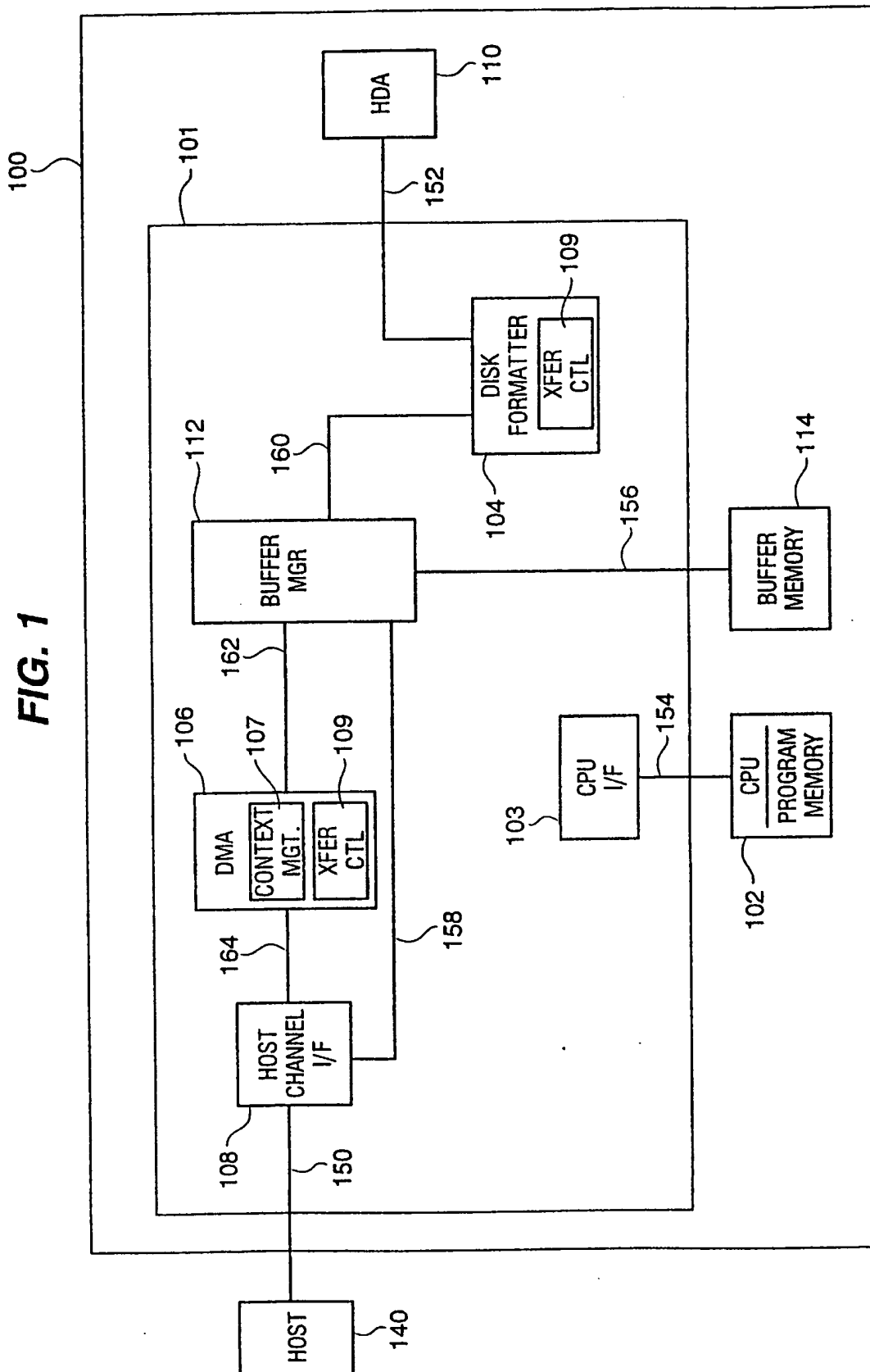
10

15. In a storage target device controller having a host channel for exchange of information with an attached host and having a disk channel for exchange of information with the storage medium, a system for processing multiple command contexts comprising:
- 15 an active context register set for storage of active parameters related to processing of an active command;
- an inactive context register set for storage of inactive parameters related to processing of an inactive command; and
- means for swapping the contents of said active context register set
- 20 with said inactive context register set.

16. The system of claim 15 further comprising:
- means for determining that said host channel is underutilized wherein said means for swapping is responsive to the determination that said host
- 25 channel is underutilized,
- wherein said means for swapping is responsive to the determination that said host channel is underutilized.

17. The system of claim 16 wherein said means for determining that said
- 30 host channel is underutilized includes:
- means for determining that said active command is presently inactive with respect to said host channel due to interactions with said disk channel.

18. The system of claim 15 further comprising:
link means in said active context register set to indicate a next one of
said multiple command contexts to be copied to said active context register
5 set following completion of processing of said active command.
19. The system of claim 18 wherein said next one of said multiple
commands is said inactive command stored in said inactive context register
set and wherein said means for swapping is responsive to said link means to
10 copy said inactive context register set to said active context register set upon
completion of processing of said active command.
20. The system of claim 15 further comprising:
means for determining that said disk channel requires additional
15 exchange of information by said host channel to continue operation of said
disk channel,
wherein said means for swapping is responsive to the determination
that said disk channel requires additional exchange of information by said
host channel to continue operation of said disk channel.



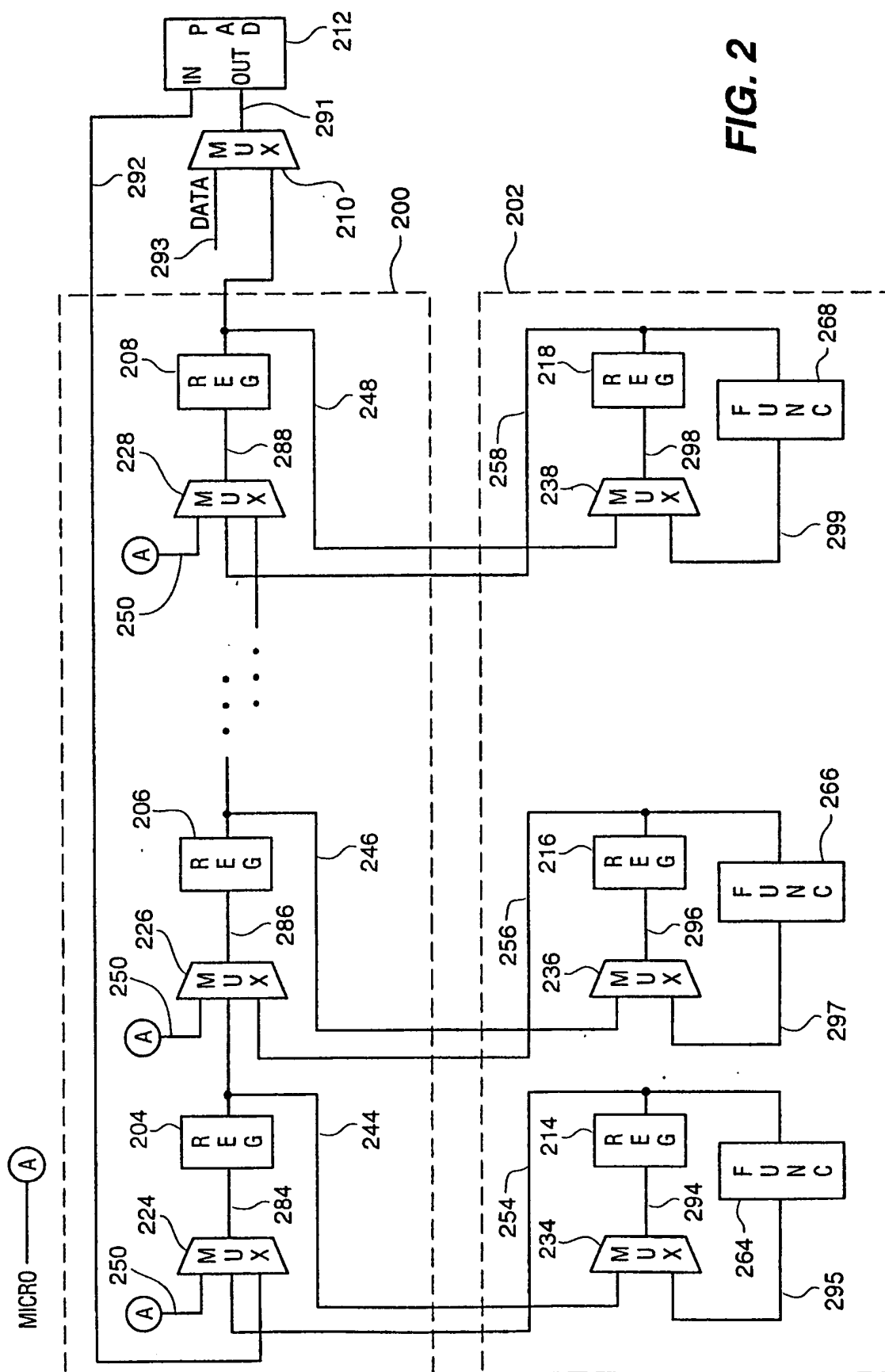


FIG. 2

FIG. 3

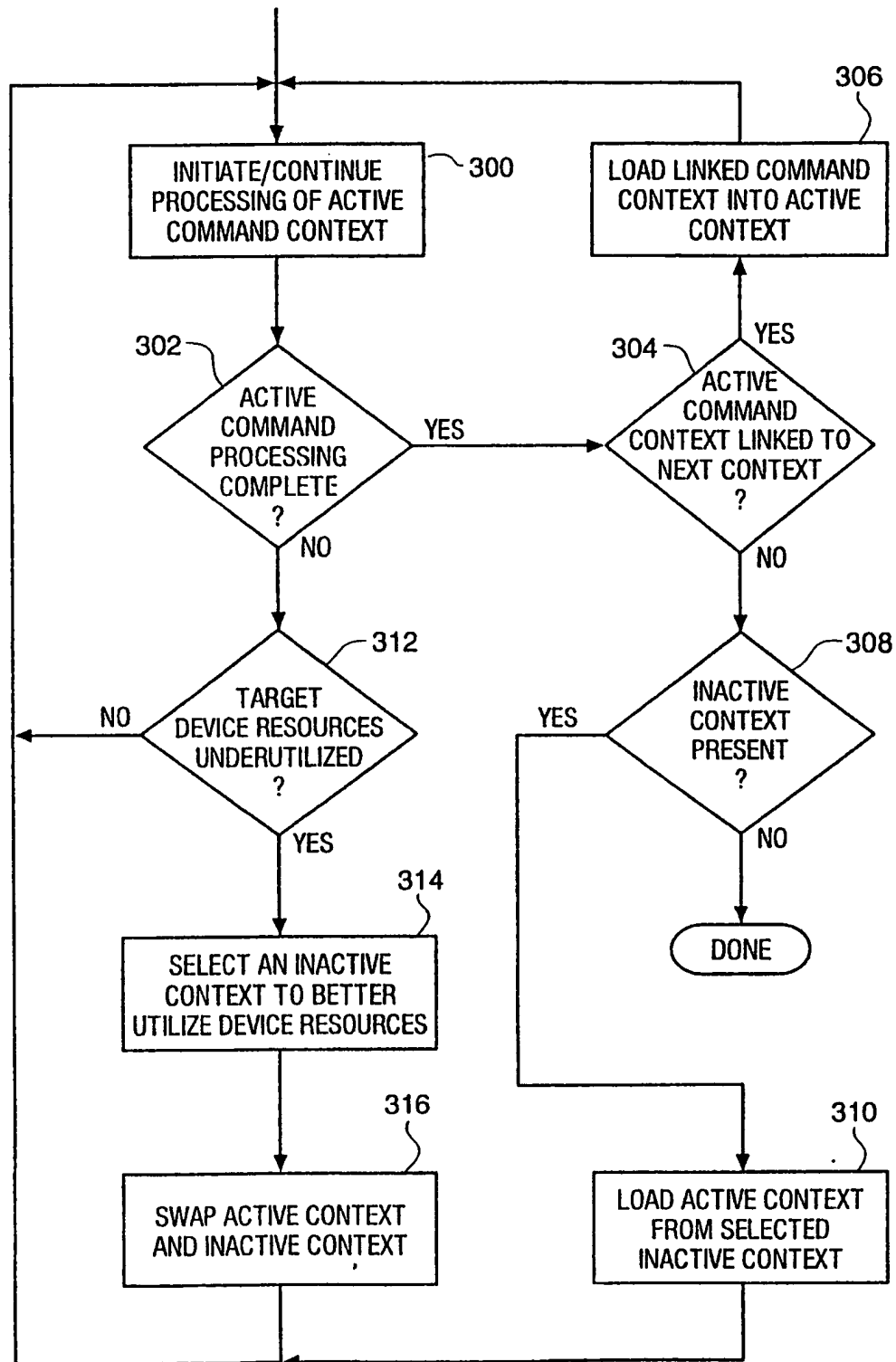


FIG. 4

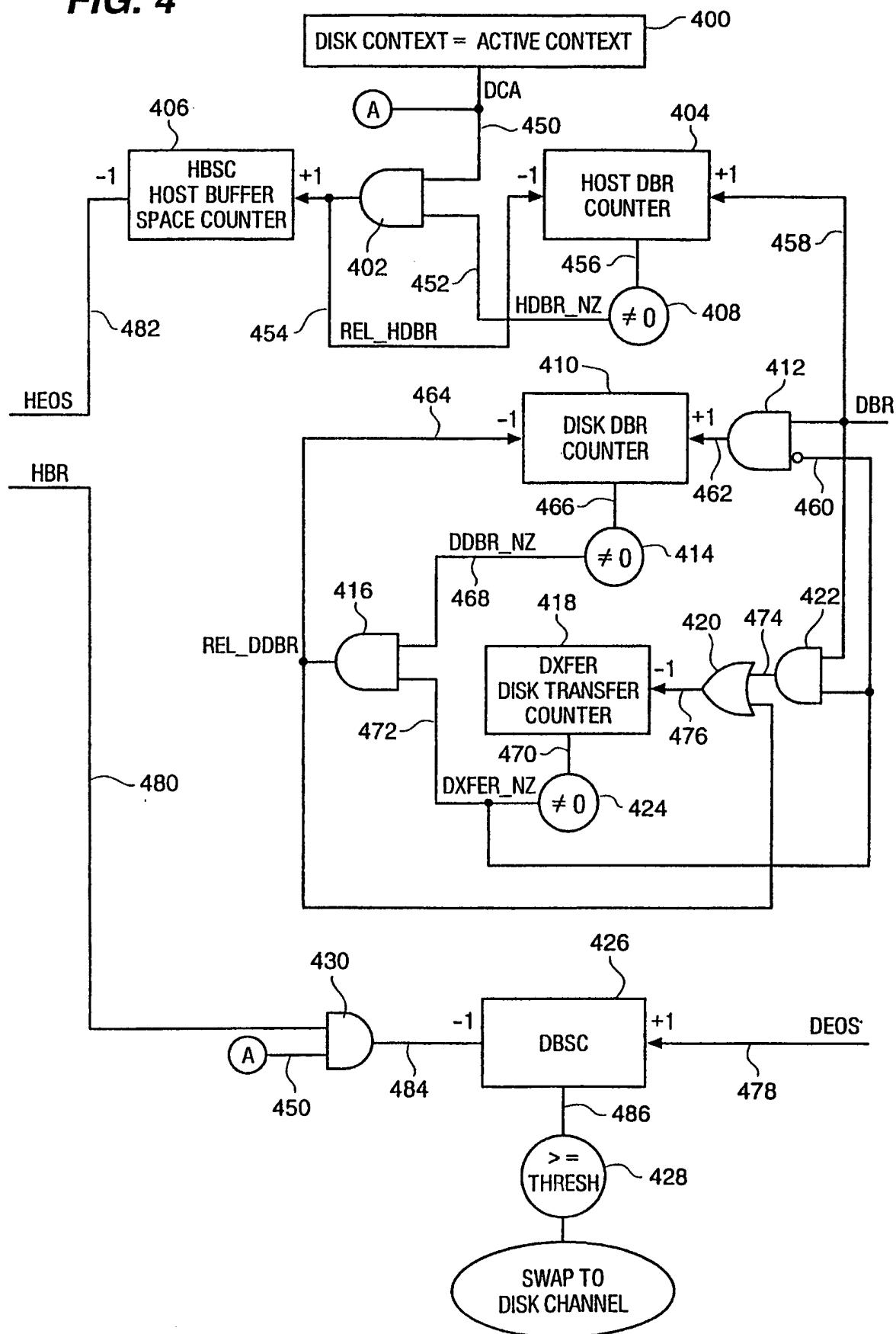
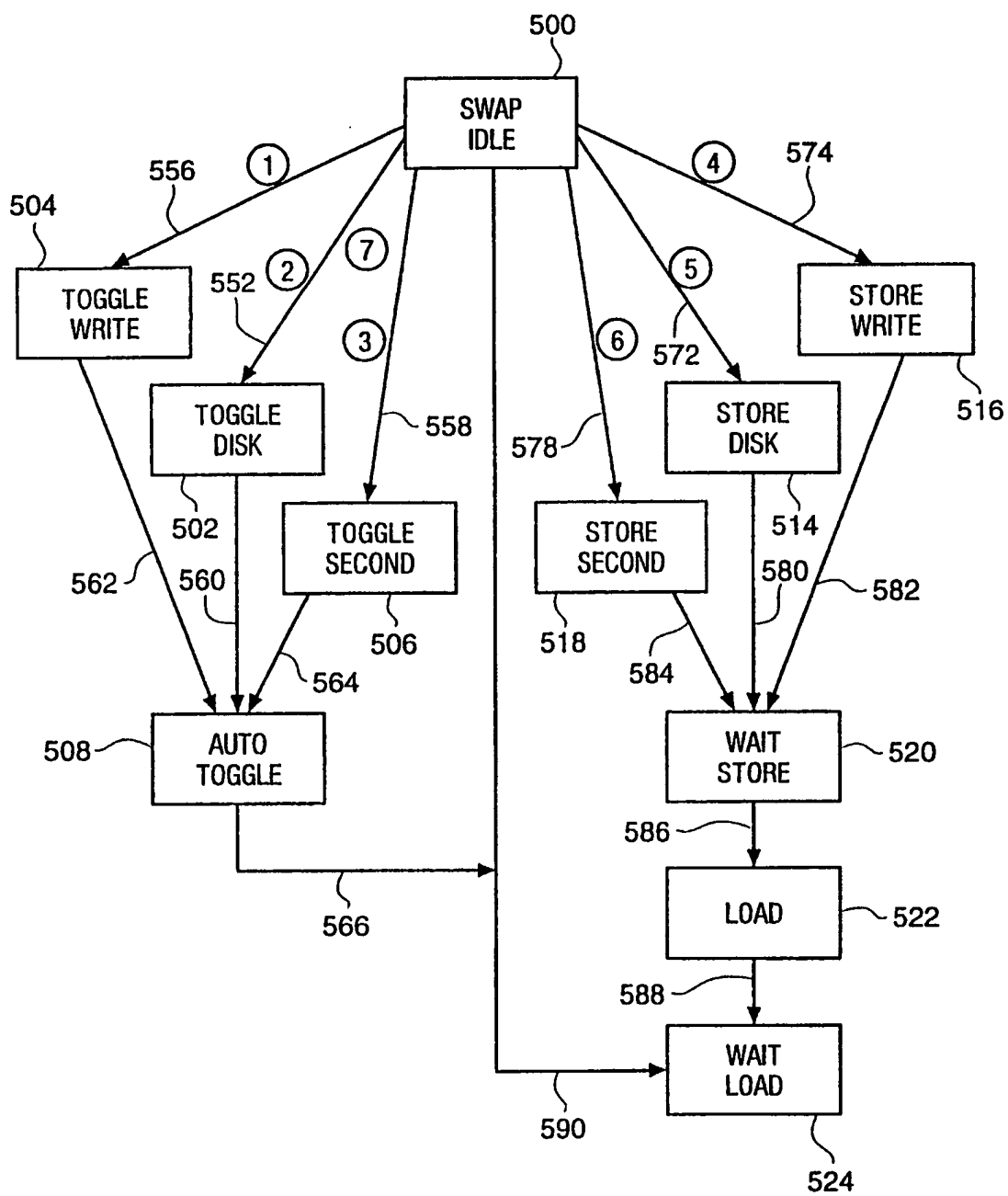


FIG. 5

INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 97/17439

A. CLASSIFICATION OF SUBJECT MATTER

IPC 6 G06F3/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 371 861 A (KEENER ET AL) 6 December 1994 see column 4, line 65 - column 6, line 60; figure 4	1, 9, 15
A	US 5 280 603 A (JEPPSON ET AL) 18 January 1994 see column 2, line 14 - column 3, line 39 see column 4, line 13 - column 6, line 26; figure 1	1, 9, 15



Further documents are listed in the continuation of box C.



Patent family members are listed in annex.

Special categories of cited documents:

- "A" document defining the general state of the art which is not considered to be of particular relevance
- "E" earlier document but published on or after the international filing date
- "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- "O" document referring to an oral disclosure, use, exhibition or other means
- "P" document published prior to the international filing date but later than the priority date claimed

- "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- "&" document member of the same patent family

Date of the actual completion of the international search

5 February 1998

Date of mailing of the international search report

13/02/1998

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040. Tx. 31 651 epo nl.
Fax: (+31-70) 340-3016

Authorized officer

Gill, S

INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 97/17439

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication where appropriate, of the relevant passages	Relevant to claim No.
A	<p>"Common Microcode Finite State Machine for Driving Both an Enhanced Small Disk Interface and an ANSI 3.8 Interface"</p> <p>IBM TECHNICAL DISCLOSURE BULLETIN., vol. 32, no. 3b, August 1989, NEW YORK US,</p> <p>pages 92-105, XP000029909</p> <p>see page 96, paragraph 1 - page 97, paragraph 4; figures 1,2</p> <p>-----</p>	1,9,15

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/US 97/17439

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5371861 A	06-12-94	JP 2507227 B JP 6187470 A	12-06-96 08-07-94
US 5280603 A	18-01-94	NONE	